UNIVERSITY OF CAMBRIDGE
Department of Engineering

Efficient decoding with continuous rational kernels using the expectation semiring

R. C. van Dalen rcv25@cam.ac.uk
A. Ragni ar527@cam.ac.uk
M. J. F. Gales mjfg@eng.cam.ac.uk

Technical Report CUED/F-INFENG/TR.674

February 2012

Abstract

Semi-Markov conditional random fields are discriminative models that can be used for speech recognition. They allow per-word (instead of per-frame) features. Since the segmentation into words is not known a priori, all possibilities must be considered. It is therefore important to consider the efficiency of the feature extraction process. Features derived from generative models like HMMs (log-likelihoods and their derivatives) allow existing adaptation methods to be used. This is equivalent to using generative kernels. Continuous rational kernels are generative kernels that represent generative likelihoods with weighted finite state transducers. This paper proposes a method to compute first- and second-order score spaces derived from HMMs for all possible segmentations in amortised constant time. It uses weighted finite state transducers with weights in a second-order *expectation semiring*.

1 Introduction

Most current speech recognition systems are based on hidden Markov models (HMMs). They therefore make a frame-level Markov assumption: given a sequence of states, the features extracted from consecutive time slices of audio are assumed independent. An alternative is to use a discriminative classifier for segments of audio, say, words, at a time (Smith and Gales 2001). A natural choice of classifier would be one that apply kernel methods, e.g. support vector machines (SVMs), which perform well for many tasks. There are three problems with applying this to speech recognition: first, a kernel must be defined between two audio segments of different length; second, the structure of speech must be used or the number of classes will be infinite; third, the segmentation of an utterance is unknown a priori.

Kernels that can handle inputs of varying length are called dynamic kernels. The choices for dynamic kernels are more limited than for kernels over fixed-length vectors. For two sequences of discrete symbols, many useful kernels can be written as rational kernels (Cortes *et al.* 2004). These can be expressed as weighted finite state transducers (WFSTs) that take one sequence as an input and another as an output. The weight that the transducer assigns to a pair of input and output defines the value of the kernel function. To extend this to sequences of continuous data, such as a segment of speech, continuous rational kernels (Layton and Gales 2007) have been proposed. They replace the deterministic input to a rational kernel by a weighted finite state automaton. This automaton can be found using from the continuous input data with a generative model, for example. To leverage the advantage of the WFST representations, however, the kernel must be written as a number of WFST compositions. This requires the transducers to be over the same semiring, which restricts the forms of kernel that can be used. This paper will instead use the primal representation and perform the computations immediately in a score space derived from one segment of audio. This restricts the form of kernel to be the inner product between score-spaces.

The second problem is that an SVM in its standard implementation is a binary classifier, whereas there exist an exponential number of possible sentences. The structured SVM (Taskar *et al.* 2003) generalises the SVM classifier to multi-class structured output labels. The form of its decision boundary is equivalent to that of a log-linear model where features are extracted per segment of data. This describes a semi-Markov model (Sarawagi and Cohen 2004) whose parameters work on the primal form of the kernel. This paper will extract features per word, but other levels of segmentation are possible. However, unlike frames, words have variable lengths. During decoding, all segmentations of the recorded utterance into words must therefore be considered. The computational efficiency then becomes an issue. This can be circumvented by using lattices from a separate speech recogniser to constrain the search space. However, a set-up with a dual system is impractical. Also, restricting segmentations to those optimal the HMM system whose limitations the semi-Markov model attempts to circumvent is a premature pessimisation. It is therefore useful to implement a feature extraction process that efficiently computes features for a range of possible segmentations.

Generative score-spaces, which generalise Fisher score-spaces (Jaakkola and Haussler 1998), consist of log-likelihoods of generative models, and their derivatives. With a zeroth-order generative score-space, which contains only the log-likelihood itself, based on word HMMs, it is possible to reproduce the exact results of an HMM speech recogniser. This allows state-of-the-art techniques for HMM speech recognisers, such as methods for noise-robustness, to be leveraged. Secondly, derivatives even of frame-level log-likelihoods are functions of all frames in the segment. Thus, the conditional independence assumptions of the HMM are relaxed.

Since the derivatives in the generative score-space depend on the frames in a whole segment, it seems obvious that they need to be re-computed completely for every hypothesised segment. This is, indeed, what Layton (2006) did, with an algorithm with nested passes of forward–backward that was essentially run separately for each hypothesised segmentation. However, this paper will introduce a method that incrementally, with only a forward pass, computes scores for all segmentations that share a start time. It will view the generative model as a weighted finite state transducer. In this formalism it is possible to generalise the weights (which would canonically represent HMM output and transition probabilities) to another semiring. This paper will use *expectation semirings*, which allow for more extensive book-keeping. As long as the HMMs have only few states, which for word HMMs is the case, this algorithm requires a modest amount of extra storage. Its advantage is that in combination with a method that finds the optimal segmentation, it computes the scores in amortised constant time.

Section 4 will introduce weighted finite state transducers and discuss how to use them to compute word likelihoods given by hidden Markov models. Section 2 will discuss continuous rational kernels based on word likelihoods for a given segmentation. Section 3 will describe the log-linear model that this paper will use. Section 4 will represent work likelihoods as weighted finite state transducers. The algorithms from Layton (2006) will be the focus of section 5. Section 6 will introduce a method that computes log-likelihoods and their derivatives for segments with different end times in one pass. Experimental results will be given in section 7.

2 Continuous rational kernels

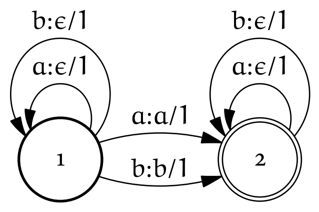
Though kernel functions often operate on fixed-length vectors, it is possible to define *dynamic kernels*, which operate on variable-length sequences. Rational kernels (Cortes *et al.* 2004) are defined on sequences of discrete symbols using weighted finite state transducers. Many useful kernels can be described as rational kernels. For example, bag-of-word kernels apply an inner product of two vectors that indicate the number of occurrences of each words. The bag-of-word kernel can be generalised to a *string kernel*, which counts all strings of a certain length.

One way of representing a bigram kernel, which counts strings of length 2, is as a composition of two transducers. Figure 1b on the facing page illustrates a bigram transducer $\mathcal{B}$ over an alphabet $\{a, b\}$, with weights in $\mathbb{N}$. Each path through this transducer indicates an occurrence of two consecutive symbols. One way of constructing a rational kernel is to compose a transducer $\mathcal{B}$ with its inverse $\mathcal{B}^{-1}$. The bigram kernel over two sequences x and y can be defined as

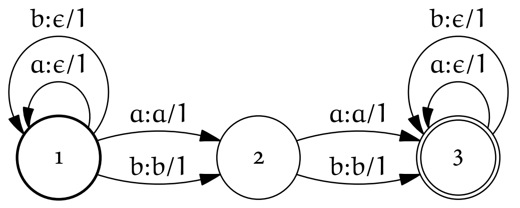
$$k(x, y) \triangleq (\mathcal{B} \circ \mathcal{B}^{-1})(x, y). \tag{1}$$

By expressing these kernels as WFST composition, which can be performed lazily (Mohri *et al.* 2000), the feature vectors do not have to be instantiated to evaluate the kernel function.

Continuous rational kernels (Layton and Gales 2007) are based on weighted finite state transducers but allow sequences of continuous-valued elements as inputs. It is



(a) Unigram transducer.



(b) Bigram transducer $\mathcal{B}$.

Figure 1 n*-gram transducers over an alphabet* $\{a, b\}$.

often possible to find a transducer $\mathcal{T}$ from a generative model. In such a transducer, each path corresponds to a latent variable sequence and weight of the path indicates the likelihood of a given observation for the state sequence. Section 4 will present such a transducer for a hidden Markov model with continuous output distributions. If the two sequences are represented with $\mathcal{T}_1$ and $\mathcal{T}_2$, then the value of the kernel can be read off from, say, the composed automaton $\mathcal{T}_1 \circ \mathcal{B} \circ \mathcal{B}^{-1} \circ \mathcal{T}_2^{-1}$.

To use the WFST-based innards of continuous rational kernels to practical, as well as theoretical, advantage, this composition must be optimised. This requires that the weights on all transducers are in the same semiring (or different semirings related by a semiring morphism). This requirement restricts the form of the kernel. The alternative, which this paper will use, is to work directly on the primal representation of the kernel. This means that the kernel is restricted to be the inner product of two vectors in these score-spaces. The next section will use the score-space representation.

3 Semi-Markov conditional random fields

A classification problem where labels have structure, such as sentences, is hard to formulate with SVMs, which are binary classifiers. Structured SVMs (SSVMs) (Taskar *et al.* 2003) are a generalisation of SVMs to multiple and structured classes. The form that classification takes is the same as classification with a log-linear conditional model. It is therefore possible to train the same type of model either as an SSVM, with a large-margin criterion, or as a log-linear conditional model, by maximising the conditional likelihood or a variant of minimum Bayes risk training. This paper will use the latter approach.

An additional issue for speech recognition, which SSVMs do not address, is that the input sequence must be segmented into, say, words. The conditional model that this yields is a semi-Markov conditional random field, which is a conditional model. This means that it models the probability of hidden variables, in this case the word sequence $\boldsymbol{w}$, given the observation sequence $\mathbf{O}$. Each of the elements w_i of $\boldsymbol{w}$ is equal to one element v from the vocabulary $\boldsymbol{v}$. The log-linear form of the model can be written:

$$P(\boldsymbol{w}|\mathbf{O}, \mathbf{s}; \boldsymbol{\alpha}) \triangleq \frac{1}{Z(\mathbf{O})} \exp\bigl(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})\bigr) . \tag{2}$$

Here, $\mathbf{s} = \{s_i\}_{i=1}^{|\boldsymbol{w}|}$ is a segmentation of the observation sequence into segments s_i. $Z(\mathbf{O})$ is the normalisation constant. $\boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s})$ is the feature function that returns a feature vector characterising the whole observation sequence. $\boldsymbol{\alpha}$ is the parameter vector. In a semi-Markov model the distribution factorises over the segments, i.e. the feature function is a sum of features for each segment:

$$\boldsymbol{\phi}(\mathbf{O}, \boldsymbol{w}, \mathbf{s}) \triangleq \sum_i \boldsymbol{\phi}(\mathbf{O}_{s_i}, w_i), \tag{3}$$

where $\mathbf{O}_{s_i}$ indicates the observations in segment s_i. In this work the feature vector is divided into separate sets of dimensions for each vocabulary entry v, and the other dimensions are zero:

$$\boldsymbol{\phi}(\mathbf{O}, w) = \begin{bmatrix} \delta(w = 1)\, \boldsymbol{\phi}_1(\mathbf{O}) \\ \vdots \\ \delta(w = V)\, \boldsymbol{\phi}_V(\mathbf{O}) \end{bmatrix}, \tag{4}$$

where $\delta(\cdot = \cdot)$ equals 1 if its argument is true, and 0 otherwise. In this expression, it selects the feature vector $\boldsymbol{\phi}_v(\mathbf{O})$ for word v.

In this work a language model was not used though approaches exist to incorporate it (Zweig and Nguyen 2010; Layton 2006). Given a segmentation, therefore, the classification of each of the segments is separate. Decoding then becomes a dynamic programming problem. The best word sequence and segmentation up to observation t can therefore be found recursively (Sarawagi and Cohen 2004; Ragni and Gales 2012):

$$\rho_t = \max_{\tau<t} \Bigl\{ \rho_\tau \max_v \bigl(\boldsymbol{\alpha}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{O}_{\tau:t-1}, w_i)\bigr) \Bigr\}, \qquad \rho_1 \triangleq 0. \tag{5}$$

ρ_T then yields the best log-unnormalised-probability for the whole utterance. The corresponding segmentation and word sequence can be found by recording the best segmentations and word for each time. Given $\rho_1 \ldots \rho_{t-1}$, evaluating (5) to find ρ_t requires $\Theta(t \cdot V)$ evaluations of $\boldsymbol{\phi}_v$. The total number of times $\boldsymbol{\phi}_v$ must be evaluated to decode $\mathbf{O}_{1:T}$ is therefore $\Theta\bigl(T^2 \cdot V\bigr)$.

In contrast, the standard Viterbi algorithm runs in $\Theta(T \cdot V)$ time. It requires the overall model to be frame-level Markov. Interpreted in terms of (5), the inner maximisation is computed in such a way that it takes $\Theta(T \cdot V)$ time. This exploits the fact that much of the computation is shared between the evaluation of $\boldsymbol{\phi}_v$ for consecutive t.

Without the frame-level Markov assumption in the model, finding the best segmentation requires at least $\Omega\bigl(T^2 \cdot V\bigr)$ evaluations of $\boldsymbol{\phi}_v$. It may be possible to reduce this by approximations, but this is not done in this initial investigation. This paper will use generative score-spaces, which break the frame-level Markov assumption. It will introduce a methods for computing $\boldsymbol{\phi}_v(\mathbf{O}_{\tau:t-1})$ in amortised constant time when it is required for all $t = \tau, \tau + 1, \ldots T$.

3.1 Generative score-spaces

The features this paper will use are generative scores (Smith and Gales 2001). These are log-likelihoods of generative models and their derivatives with respect to its parameters. Thus, zeroth-, first-, and second-order score-spaces are defined as

$$\phi_\nu^{(0)}(\mathbf{O}) \triangleq \left[\, \log g(\mathbf{O}|\nu;\boldsymbol{\lambda}) \,\right]; \tag{6a}$$

$$\phi_\nu^{(1)}(\mathbf{O}) \triangleq \begin{bmatrix} \log g(\mathbf{O}|\nu;\boldsymbol{\lambda}) \\ \nabla_{\boldsymbol{\lambda}} \log g(\mathbf{O}|\nu;\boldsymbol{\lambda}) \end{bmatrix}; \tag{6b}$$

$$\phi_\nu^{(2)}(\mathbf{O}) \triangleq \begin{bmatrix} \log g(\mathbf{O}|\nu;\boldsymbol{\lambda}) \\ \nabla_{\boldsymbol{\lambda}} \log g(\mathbf{O}|\nu;\boldsymbol{\lambda}) \\ \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}} \log g(\mathbf{O}|\nu;\boldsymbol{\lambda}) \end{bmatrix}, \tag{6c}$$

where $g(\mathbf{O}|\nu;\boldsymbol{\lambda})$ is the likelihood that the generative model assigns to observation segment $\mathbf{O}$ for word ν and parameter vector $\boldsymbol{\lambda}$. The generative models in this paper will be assumed word-specific hidden Markov models, but any form of likelihood that can be expressed as a WFST can be used. If the values on the arcs are derived from a limited number of frames, the likelihoods for $o_{\tau:t}$ with fixed τ and consecutive t can be computed in one pass. Section 4 will demonstrate this. Section 6 will introduce a method that generalises this to scores that include derivatives.

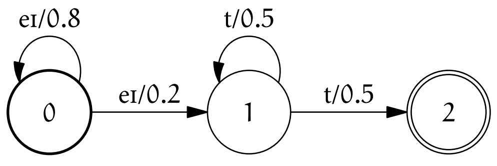
4 Word likelihoods as weighted finite state transducers

This section will show why it is useful to describe the likelihood computation of a generative model that uses a latent state sequence as as a weighted finite state transducer. This makes it possible to compute the likelihoods for $o_{\tau:t}$ with fixed τ and consecutive t in one pass. An equivalent result is presented in Ragni and Gales (2012) but expressed in terms of hidden Markov models.

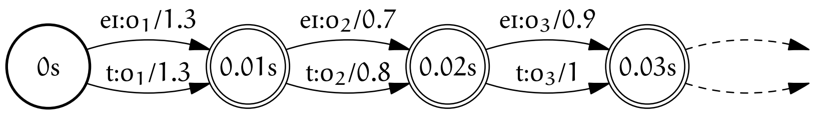
Most of the training and decoding process in an HMM-based speech recogniser can be described in terms of weighted finite state transducers (WFSTs) (for a recent overview, see Hoffmeister *et al.* 2012). This has the advantage over an HMM representation that generic algorithms and properties of transducers can be used. Specifically, algorithms for composition and finding the best path are useful, and the flexibility in choosing the domain of the weights. This section will present how to compute likelihoods for an HMM, say, for a word. This uses one automaton that represents phone sequences, and one that represents the observation.

Figure 2 illustrates the two component state machines required for computing the likelihood of a segment of speech. The automaton $\mathcal{S}$, in figure 2a, produces possible sequences of phones. It produces a sequence of one or more symbols eɪ and one or more t. A sequence of output symbols corresponds to a path, a sequence of arcs, starting at the start state ("0", with a bold circle), and finishing at the final state ("2", with a double circle). The total weight of the sequence is found by multiplying the weights of the arcs on the path. This represents the probability of the phone sequence. In the transducer drawn here, the probability is normalised, but that is not required of WFSTs.

For segments starting at each time, a transducer is set up representing phone likelihoods for the observations. A weighted finite state transducer $\mathcal{O}$ that does that for $\mathbf{O}_{1:t}$ for consecutive t is depicted in figure 2b. Each state here represents a time (indicated in fractions of seconds from 0s). The audio in between two consecutive time steps is



(a) Weighted automaton $\mathcal{S}$ produces a sequenceof phones.



(b) WFST $\mathcal{O}$ with phone likelihoods for each observation o_t.

Figure 2 *Component WFSTs for computing the likelihood of a segment.*

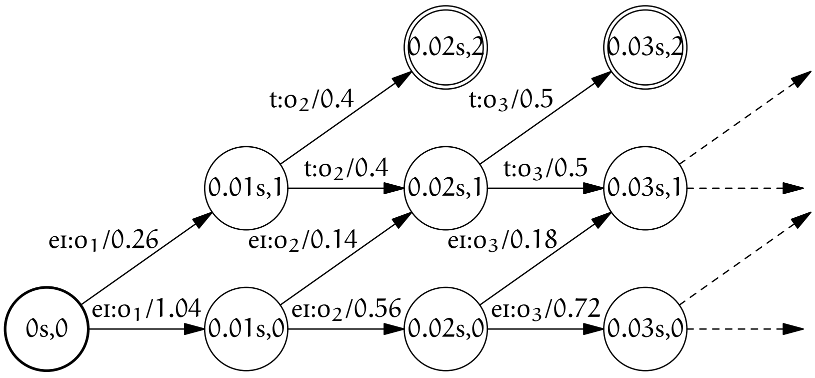


Figure 3 *WFST $\mathcal{T}$ for computing the likelihood of a segment: the two WFSTs from figure 2 composed.*

represented by feature vector o_t. For each observation there is an arc for each of the phones. The weight on each arc is set to the likelihood that the output distribution of that phone (say, a mixture of Gaussians) gives for that observation o_t. The product of the weights on the arcs on a path through this transducer therefore equals the observation likelihood for the phone sequence corresponding to the path.

In theory, automaton $\mathcal{S}$ produces an infinite number of sequences of phones with a weight assigned to each of the sequences. Transducer $\mathcal{O}$ represents an exponential number of pairs of an input and an output sequence, where each input sequence consists of phonemes, and each output sequence of observations $o_1, o_2, \ldots$. The interest is in only those combinations of sequences from $\mathcal{S}$ and $\mathcal{O}$ with the same phone sequence. Because both automata are finite-state machines, it is possible to represent the sequences of interest, and their weights, with a third transducer $\mathcal{T}$, which is obtained through composition of $\mathcal{S}$ and $\mathcal{O}$:

$$\mathcal{T} = \mathcal{S} \circ \mathcal{O}. \tag{7}$$

This yields the transducer in figure 3. The states of this transducer are pairs of states from the two original transducers. For clarity, the states corresponding to one state from the transducer $\mathcal{O}$ representing the observations are in the same horizontal position, and those corresponding to phone sequences in $\mathcal{S}$ in the same vertical position. This way, the graph corresponds to the trellis diagrams sometimes drawn to explain Viterbi or the forward algorithm in HMMs.

The product of the weights that $\mathcal{S}$ and $\mathcal{O}$ assign represents the joint distribution of phone sequence and observation sequence. This is equivalent to the product of the weights on the corresponding path $\pi = e_1 \ldots e_t$ from a start to an end state in the WFST $\mathcal{T}$. This makes it possible to perform operations on the WFST that have probabilistic interpretations. Denote the weight of arc e with $l[e; \lambda]$, its start node with $p[e]$ and its end node with $n[e]$. The highest likelihood corresponding to one path leading to node q can be computed recursively as

$$\text{best}(q) \triangleq \begin{cases} 1, & \text{if } q \text{ an initial state;} \\ \max_{e:n[e]=q} l[e;\lambda]\, \text{best}(p[e]), & \text{otherwise.} \end{cases} \tag{8}$$

The “Viterbi” algorithm is a time-synchronous algorithm to compute these values. An interesting aspect of WFSTs is that this function can be generalised by generalising the operations used to combine consecutive arcs and of competing paths. Denoting the operation that combines the weights of consecutive arcs in one path with $\otimes$, and the one that combines the weights of different paths with $\oplus$, and the multiplicative identity with $\bar{1}$, the definition in (8) becomes

$$\text{best}(q) \triangleq \begin{cases} \bar{1}, & \text{if } q \text{ an initial state;} \\ \bigoplus_{e:n[e]=q} l[e;\lambda] \otimes \text{best}(p[e]), & \text{otherwise.} \end{cases} \tag{9}$$

Assuming that $\otimes$ distributes over $\oplus$, so that $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$, the recursive function can be unrolled to obtain

$$\text{best}(q, \mathcal{T}) = \bigoplus_{\pi:n[\pi]=q} \bigotimes_{e\in\pi} l[e;\lambda] = \bigoplus_{\pi:n[\pi]=q} l[\pi;\lambda]\,, \tag{10}$$

where $\pi = e_1, \ldots, e_i$ is a path, $n[\pi] \triangleq n[e_i]$, and $l[\pi;\lambda] \triangleq \bigotimes_{e\in\pi} l[e;\lambda]$. The weight can be thought of as being in a *semiring*, which defines a set of values, operations $\oplus$, $\otimes$ (which is distributive over $\oplus$), and constants $\bar{0}$ and $\bar{1}$.

By defining $\otimes$ as $\times$ and $\oplus$ as $+$, the algorithm in (10) yields the sum of the weights of all paths. It is then called the “forward” algorithm. Applying it to a final node q yields the likelihood of a corresponding observation sequence $\mathbf{O}$:

$$\text{forward}(q, \mathcal{T}) = \sum_{\pi:n[\pi]=q} l[\pi;\lambda]\,. \tag{11}$$

The backward algorithm can be defined as the forward algorithm on the same transducer with all arcs reversed. This reversed transducer will be denoted with $\mathcal{T}^{\mathsf{T}}$. Then,

$$\text{backward}(q, \mathcal{T}) \triangleq \text{forward}(q, \mathcal{T}^{\mathsf{T}}). \tag{12}$$

This algorithm takes $\Theta(|\mathcal{T}|)$ time, where $|\mathcal{T}|$ is the number of arcs in $\mathcal{T}$. It works without modification or extra cost on a transducer with multiple final states, such as $\mathcal{T}$ in figure 3. it then yields the likelihood for each of the final states.

$\phi_v^{(0)}(\mathbf{O}_{\tau:t})$ can then be computed for $t = \tau + 1, \ldots, T$ as follows. Produce a WFSA $\mathcal{S}$ for word v like in figure 2a. It is assumed that the number of states in $\mathcal{S}$ is constant. Represent observations $\tau : T$ with WFST $\mathcal{O}$ like in figure 2b. Running the forward algorithm on the composition of these transducers will produce the likelihood for consecutive $t = \tau + 1, \ldots, T$ in the end states of the transducer in amortised constant time. The zeroth-order generative score is found as the logarithm of the likelihood:

$$\log l(\boldsymbol{\lambda}) = \log \sum_{\pi} l[\pi; \boldsymbol{\lambda}] . \tag{13}$$

If the transducer contains multiple start states, then their likelihoods would be summed. In the context of this paper, that is not desirable: in (5) values of $\boldsymbol{\phi}(\mathbf{O}_{\tau:t-1}, w_i)$ for different τ are compared, not summed.[1] The algorithm therefore needs to be run separately for each start time. However, since $\phi_v^{(0)}(\mathbf{O}_{\tau:t})$ must be evaluated separately for each combination of τ and t, the time needed per evaluation is still $\Theta(1)$. This result from Ragni and Gales (2012) will be extended to higher-order generative score-spaces in section 6.

5 Computing generative score-spaces explicitly

Apart from log-likelihoods, generative score-space also contain their derivatives. Layton (2006) introduced an explicit method for computing these, which this section will discuss. Section 6 will introduce a faster method using expectation semirings.

The first method for computing the generative scores expresses the derivatives in terms of the posterior of the hidden parameters (Layton 2006). This is possible the derivatives in the generative score-spaces consider infinitesimal changes to the parameters. Finding the posterior of the hidden parameters (here, the arc posterior suffices), is a well-known operation from standard speech recogniser training, where the distribution over the hidden variables is explicitly kept constant while optimising the parameters of the model. As in section 4, the likelihood of the generative model can be expressed with a weighted finite state transducer, with weights l, which will be written as functions of the generative model: $l[e; \boldsymbol{\lambda}]$. The likelihood is equal to the sum of the weights of each path in WFST $\mathcal{T}$:

$$l(\boldsymbol{\lambda}) = \sum_{\pi} l[\pi; \boldsymbol{\lambda}] ; \qquad\qquad l[\pi; \boldsymbol{\lambda}] = \prod_{e \in \pi} l[e; \boldsymbol{\lambda}] \tag{14}$$

The zeroth-order score is defined as the log-likelihood given by the generative model, computed as in (13) Higher-order scores are defined by the derivatives with respect to the generative parameters. The derivations of these are given in appendix A.

$$\nabla_{\boldsymbol{\lambda}} \log l(\boldsymbol{\lambda}) = \sum_{\pi} \frac{l[\pi; \boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi'; \boldsymbol{\lambda}]} \sum_{e \in \pi} \nabla_{\boldsymbol{\lambda}} \log l[e; \boldsymbol{\lambda}] ; \tag{15a}$$

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}} \log l(\boldsymbol{\lambda}) = \sum_{\pi} \frac{l[\pi; \boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi'; \boldsymbol{\lambda}]} &\left(\left(\sum_{e \in \pi} \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \log l[e; \boldsymbol{\lambda}] \right) \left(\sum_{e \in \pi} \nabla_{\boldsymbol{\lambda}} \log l[e; \boldsymbol{\lambda}] \right) \right. \\ &\left. + \sum_{e \in \pi} \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}} \log l[e; \boldsymbol{\lambda}] \right) \\ &- \left(\nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \log l(\boldsymbol{\lambda}) \right) \left(\nabla_{\boldsymbol{\lambda}} \log l(\boldsymbol{\lambda}) \right) . \end{aligned} \tag{15b}$$

[1] If $\oplus$ is defined as max, then it is possible to have multiple initial states (with weight ρ_{τ}), which recovers the frame-level Viterbi algorithm.

For first-order scores, the summations in (15a) can be reversed, factoring out a term related to the fraction of the total weight going through one arc:

$$\begin{aligned}\nabla_\lambda \log l(\lambda) &= \sum_\pi \frac{l[\pi;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} \sum_{e\in\pi} \nabla_\lambda \log l[e;\lambda_j] \\ &= \sum_e (\nabla_\lambda \log l[e;\lambda]) \frac{\sum_{\pi:e\in\pi} l[\pi;\lambda]}{\sum_\pi l[\pi;\lambda]} \\ &= \sum_e \gamma_e \nabla_\lambda \log l[e;\lambda]\,, \end{aligned} \tag{16a}$$

where the arc posterior is defined as

$$\gamma_e \triangleq \frac{\sum_{\pi:e\in\pi} l[\pi;\lambda]}{\sum_\pi l[\pi;\lambda]}. \tag{16b}$$

The arc posteriors can be computed with the forward–backward algorithm. This uses the forward and backward scores to compute the numerator in (16b). For an arc e,

$$\begin{aligned}&\text{forward}(p[e], \mathcal{T}) \cdot l[e;\lambda] \cdot \text{backward}(n[e], \mathcal{T}) \\ &= \Bigg(\sum_{\pi:n[\pi]=p[e]} l[\pi;\lambda]\Bigg) \cdot l[e;\lambda] \cdot \Bigg(\sum_{\pi:p[\pi]=n[e]} l[\pi;\lambda]\Bigg) \\ &= \sum_{\pi:e\in\pi} l[\pi;\lambda]\,. \end{aligned} \tag{17}$$

This gives the numerator of γ_e. It can be normalised across all time-synchronous arcs in a transducer like in figure 3, or by dividing by the forward score in the final state. Since the forward probabilities are shared between $\phi_v^{(1)}(\mathbf{O}_{\tau:t})$ with the same τ, and similar for the backward probabilities, it takes only $\Theta(T^2)$ to find all forward and backward probabilities. However, (17) must be evaluated separately for each combination of τ and t, so that the overall process of gathering statistics in (16a) takes $\Theta(T^3)$ time (Ragni and Gales 2012).

Second-order scores can be expressed in terms of double arc posteriors $\gamma_{ee'}$:

$$\begin{aligned}&\nabla_\lambda^T \nabla_\lambda \log l(\lambda) + (\nabla_\lambda^T \log l(\lambda))(\nabla_\lambda \log l(\lambda)) \\ &= \sum_\pi \frac{l[\pi;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} \left(\sum_{e\in\pi} \nabla_\lambda^T \log l[e;\lambda] \sum_{e'\in\pi} \nabla_\lambda \log l[e';\lambda] + \sum_{e\in\pi} \nabla_\lambda^T \nabla_\lambda \log l[e;\lambda]\right) \\ &= \sum_{e,e'} \nabla_\lambda^T \log l[e;\lambda]\, \nabla_\lambda \log l[e';\lambda] \frac{\sum_{\pi:\{e,e'\}\subseteq\pi} l[\pi;\lambda]}{\sum_\pi l[\pi;\lambda]} \\ &\quad + \sum_e \nabla_\lambda^T \nabla_\lambda \log l[e;\lambda] \frac{\sum_{\pi:e\in\pi} l[\pi;\lambda]}{\sum_\pi l[\pi;\lambda]} \\ &= \sum_{e,e'} \gamma_{ee'} \nabla_\lambda^T \log l[e;\lambda]\, \nabla_\lambda \log l[e';\lambda] + \sum_e \gamma_e \nabla_\lambda^T \nabla_\lambda \log l[e;\lambda]\,, \end{aligned} \tag{18}$$

where

$$\gamma_{ee'} \triangleq \frac{\sum_{\pi:\{e,e'\}\subseteq\pi} l[\pi;\lambda]}{\sum_\pi l[\pi;\lambda]}. \tag{19}$$

Layton (2006) introduced an algorithm for computing this that would take $\Theta(T^5)$ for an utterance. This is impractical. The next section will therefore introduce a method that finds generative scores in $\Theta(T^2)$ time overall.

6 Computing scores with higher-order expectation semirings

The *expectation semiring* was introduced in Eisner (2002). Its initial purpose was to allow expectation–maximisation on weighted finite state transducers with a probabilistic interpretation. The statistics, which in speech recognition training would be computed after applying the forward–backward algorithm, are appended to the weights. By defining the weights to be in the expectation semiring, which defines operations $\oplus$ and $\otimes$ in a specific way, only a forward pass is required to gather all required statistics. For normal speech recognition training, the cost of carrying statistics for all sub-phones in the expanded HMM in each state would be prohibitive. However, in this paper the HMMs are small, and different lengths for the observation segments need to be considered. The following will therefore define weights in a semiring so that the algorithm in section 4 can be applied, and higher-order generative scores computed in amortised constant time.

A simple way of viewing the required semiring is as appending derivatives to the weights (Li and Eisner 2009). The weight for arcs e then becomes

$$w[e;\lambda] \triangleq \langle l[e;\lambda]\,, \nabla_\lambda l[e;\lambda]\rangle\,. \tag{20}$$

The semiring operations defined on these new weights can be derived in various ways. The simplest for the purpose of this paper is to describe the derivatives of the sum or product of two weights l_1 and l_2:

$$\nabla_\lambda(l_1 + l_2) = \nabla_\lambda l_1 + \nabla_\lambda l_2; \tag{21a}$$
$$\nabla_\lambda(l_1 \cdot l_2) = l_1 \cdot \nabla_\lambda l_2 + l_2 \cdot \nabla_\lambda l_1. \tag{21b}$$

Denoting the weights with $\langle l, l'\rangle$, the semiring operations should be defined as

$$\langle l_1, l'_1\rangle \oplus \langle l_2, l'_2\rangle \triangleq \langle l_1 + l_2, l'_1 + l'_2\rangle\,; \tag{22a}$$
$$\langle l_1, l'_1\rangle \otimes \langle l_2, l'_2\rangle \triangleq \langle l_1 \cdot l_2, l_1 \cdot l'_2 + l_2 \cdot l'_1\rangle\,; \tag{22b}$$
$$\bar{0} \triangleq \langle 0, 0\rangle\,; \tag{22c}$$
$$\bar{1} \triangleq \langle 1, 0\rangle\,. \tag{22d}$$

To demonstrate that this produces the path weight and its derivative for longer paths, consider the weight of one path π starting with arc e_1. $w[\pi;\lambda]$ can then be unrolled with

$$\begin{aligned} w[\pi;\lambda] &= \bar{1} \otimes \bigotimes_{e\in\pi} w[e;\lambda] \\ &= \langle 1, 0\rangle \otimes w[e_1;\lambda] \otimes w[\pi \setminus e_1;\lambda] \\ &= \big\langle l[e_1;\lambda] \cdot l[\pi \setminus e_1;\lambda]\,,\ l[\pi \setminus e_1;\lambda] \cdot \nabla_\lambda l[e_1;\lambda] + l[e_1;\lambda] \cdot \nabla_\lambda l[\pi \setminus e_1;\lambda]\,\big\rangle \\ &= \Big\langle l[\pi;\lambda]\,,\ l[\pi;\lambda] \cdot \frac{\nabla_\lambda l[e_1;\lambda]}{l[e_1;\lambda]\,\cdot} + l[e_1;\lambda] \cdot \nabla_\lambda l[\pi \setminus e_1;\lambda]\,\Big\rangle \\ &= \left\langle l[\pi;\lambda]\,,\ l[\pi;\lambda] \sum_{e\in\pi} \frac{\nabla_\lambda l[e;\lambda]}{l[e;\lambda]} \right\rangle, \end{aligned} \tag{23}$$

which is exactly the result in (30). An intuitive way of viewing the difference with using the forward—backward algorithm, as in section 5 is the following. The forward–backward algorithm computes the arc posteriors first, by summing over all paths going in and coming out of arcs. The arc statistics are then multiplied by the arc posteriors. The forward algorithm in combination with an expectation semiring, on the other hand, finds the weights on the paths going into arcs first. Then the statistics are multiplied and taken along all paths coming out of the arc. They implicitly get post-multiplied by the weights of all these paths.

It is trivial to see that the sum over all paths of $w[\pi; \boldsymbol{\lambda}]$ gives the likelihood. Applying the forward algorithm will therefore yield in the final state for time t the values

$$\langle l(\boldsymbol{\lambda}) \,, \nabla_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda}) \rangle \,. \tag{24}$$

Finding the first-order score with the derivative of the log-likelihood (as opposed to the likelihood) is straightforward:

$$\nabla_{\boldsymbol{\lambda}} \log l(\boldsymbol{\lambda}) = \frac{\nabla_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda})}{l(\boldsymbol{\lambda})}, \tag{25}$$

which can be found with the values in (24).

It is also possible to find second-order derivatives in the same way. The *second-order expectation semiring* (Li and Eisner 2009) is found through a “lifting trick”: since first-order weights are in a semiring, their derivatives can be appended:

$$\begin{aligned} w[e; \boldsymbol{\lambda}] &\triangleq \Big\langle \big\langle l[e; \boldsymbol{\lambda}] \,, \nabla_{\boldsymbol{\lambda}} l[e; \boldsymbol{\lambda}] \,\big\rangle, \; \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \big\langle l[e; \boldsymbol{\lambda}] \,, \nabla_{\boldsymbol{\lambda}} l[e; \boldsymbol{\lambda}] \,\big\rangle \Big\rangle \\ &= \Big\langle \big\langle l[e; \boldsymbol{\lambda}] \,, \nabla_{\boldsymbol{\lambda}} l[e; \boldsymbol{\lambda}] \,\big\rangle, \; \big\langle \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} l[e; \boldsymbol{\lambda}] \,, \nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} \nabla_{\boldsymbol{\lambda}} l[e; \boldsymbol{\lambda}] \,\big\rangle \Big\rangle. \end{aligned} \tag{26a}$$

It is possible to use a different set of parameters from $\boldsymbol{\lambda}$ for the second derivative, so that it can be useful to include $\nabla_{\boldsymbol{\lambda}} l[e; \boldsymbol{\lambda}]$ and $\nabla_{\boldsymbol{\lambda}}^{\mathsf{T}} l[e; \boldsymbol{\lambda}]$. The semiring operations for the second-order expectation semiring are defined as

$$\begin{aligned} &\langle\langle l_1, l_1' \rangle, \langle l_1'', l_1''' \rangle\rangle \oplus \langle\langle l_2, l_2' \rangle, \langle l_2'', l_2''' \rangle\rangle \\ &\triangleq \langle\langle l_1, l_1' \rangle \oplus \langle l_2, l_2' \rangle, \langle l_1'', l_1''' \rangle \oplus \langle l_2'', l_2''' \rangle\rangle \\ &= \langle\langle l_1 + l_2, l_1' + l_2' \rangle, \langle l_1'' + l_2'', l_1''' + l_2''' \rangle\rangle\,; \end{aligned} \tag{26b}$$

$$\begin{aligned} &\langle\langle l_1, l_1' \rangle, \langle l_1'', l_1''' \rangle\rangle \otimes \langle\langle l_2, l_2' \rangle, \langle l_2'', l_2''' \rangle\rangle \\ &\triangleq \big\langle \langle l_1, l_1' \rangle \otimes \langle l_2, l_2' \rangle, \; \langle l_1, l_1' \rangle \otimes \langle l_2'', l_2''' \rangle + \langle l_2, l_2' \rangle \otimes \langle l_1'', l_1''' \rangle \,\big\rangle \\ &= \big\langle \langle l_1 \cdot l_2, l_1 \cdot l_2' + l_2 \cdot l_1' \rangle, \; \langle l_1 \cdot l_2'', l_1 \cdot l_2''' + l_1' \cdot l_2'' \rangle \oplus \langle l_2 \cdot l_1'', l_2 \cdot l_1''' + l_2' \cdot l_1'' \rangle \,\big\rangle \\ &= \big\langle \langle l_1 \cdot l_2, l_1 \cdot l_2' + l_2 \cdot l_1' \rangle, \; \langle l_1 \cdot l_2'' + l_2 \cdot l_1'', l_1 \cdot l_2''' + l_1' \cdot l_2'' + l_2 \cdot l_1''' + l_2' \cdot l_1'' \rangle \,\big\rangle; \end{aligned} \tag{26c}$$

$$\overline{0} \triangleq \langle 0, 0, 0, 0 \rangle\,; \tag{26d}$$

$$\overline{1} \triangleq \langle 1, 0, 0, 0 \rangle\,. \tag{26e}$$

Here, multiplication must be suitably interpreted. For example, $l_2' \cdot l_1''$ is the cross product between two vectors. This can be seen by considering derivatives with respect to different parameters.

To find the generative score, the second derivative of the log-likelihood is found with

$$\begin{aligned}\nabla_\lambda^{\mathsf{T}}\nabla_\lambda \log l(\lambda) &= \nabla_\lambda^{\mathsf{T}}\frac{\nabla_\lambda l(\lambda)}{l(\lambda)} \\ &= \frac{\nabla_\lambda^{\mathsf{T}}\nabla_\lambda l(\lambda)}{l(\lambda)} + \frac{1}{(l(\lambda))^2}(\nabla_\lambda^{\mathsf{T}} l(\lambda))(\nabla_\lambda l(\lambda)) \\ &= \frac{\nabla_\lambda^{\mathsf{T}}\nabla_\lambda l(\lambda)}{l(\lambda)} + (\nabla_\lambda^{\mathsf{T}} \log l(\lambda))(\nabla_\lambda \log l(\lambda)). \qquad (27)\end{aligned}$$

Thus, assuming the number of generative parameters constant, first- and second-order generative score-spaces can be found in amortised constant time while performing optimal decoding. Note that nothing in this section has relied on a specific meaning of the original arc weights $l[e;\lambda]$. Derivatives of another quantity than the log-likelihood that can be expressed in terms of a WFST in a similar way can also be computed with the expectation semiring.

7 Experiments

Optimal decoding with generative score-spaces was tested on a small, noisy corpus: AURORA 2. This makes it possible to test the interaction with noise compensation methods. The task uses a small vocabulary and no language model, which makes experiments without such optimisations as pruning possible. AURORA 2 (Hirsch and Pearce 2000) is a standard digit string recognition task. The generative model has whole-word HMMs with 16 states and 3 components per state. The number of HMM parameters is 46 732. The HMMs are compensated with unsupervised vector Taylor series (VTS) compensation as in Gales and Flego (2010). Three different sets of HMM parameters are used to derive features for the discriminative model: trained on clean data, trained on corrupted data with VTS adaptive training (VAT), and on corrupted data with discriminative VTS adaptive training (DVAT).

With zeroth-order score-spaces, the discriminative model has 13 parameters, corresponding to the log-likelihoods of the 13 words (11 digits plus “sil” and “sp”), found as in section 4. In first-order score-spaces the derivatives of the log-likelihood are computed as in section 6 and appended. Only derivatives of the compensated means are used, since including variances led to rapid over-fitting. The number of parameters was 21 554. Second-order score-spaces resulted in generalisation problems because of the small training set, and initial experiments did not yield improvements over first-order score-spaces.

The discriminative models were initialised to use the likelihoods from the generative models unchanged. They were then trained with a minimum Bayes risk criterion as in Ragni and Gales (2011). This used a large lattice with many, but not all, segmentations to represent the numerator and denominator. One of the three test sets, test set A, was used as the validation set to stop training.

Table 1 contains word error rates for the experiments. Comparing the first two rows of each block will give an insight in the properties of the log-linear model. The differences between second and third rows of each block indicate the effects of using derivatives as features.

Generative model	Score-space order	Test set A	B	C	Average
	—	9.8	9.1	9.5	9.5
VTS	zeroth	7.8	7.3	8.0	7.6
	first	6.8	6.4	7.3	6.7
	—	8.9	8.3	8.8	8.6
VAT	zeroth	7.1	6.8	7.5	7.1
	first	6.2	6.1	6.8	6.3
	—	6.7	6.6	7.0	6.7
DVAT	zeroth	6.6	6.5	6.9	6.6
	first	6.1	6.1	6.6	6.2

Table 1 *Word error rates for decoding with generative score-spaces.*

Results obtained by the generative model with Viterbi decoding are in the first row of each block. For the second row, log-likelihoods are extracted from this model as features for the log-linear model, and the optimal segmentation is found. For generative models with ("VTS') and without ("VAT") adaptive training this gives an improvement close to 20 % relative. However, discriminatively trained HMMS ("DVAT") are similar to the log-linear model derived from just log-likelihoods (Heigold *et al.* 2011). The most significant differences here are that the log-linear model chooses the optimal word sequence and marginalises out over state sequences within the word. This gives only a tiny improvement.

The bottom row of each block contains word error rates using first-order derivatives of log-likelihoods as features. These break the Markov assumption of HMMS. Interestingly, the effect of this seems only partly dependent of how good the underlying HMM is. The improvement compared to score-spaces with just log-likelihoods is 11–12 % relative for VTS and VAT. The discriminatively trained HMM (DVAT) has been optimised for decoding with it, rather than for use within a log-linear model. It is therefore encouraging that the relative gain with first-order derivatives is as high as 6 %.

8 Conclusion

This paper has discussed a strategy for decoding with a segmental log-linear model with features in generative score-spaces. Since the optimal segmentation of the utterance into segments is sought, the score that the model assigns for a word must be computed separately for every possible segment, i.e. $\Theta(T^2)$ times. Computing scores for a range of segments at once can then, surprisingly, be done in amortised constant time. For log-likelihood score-spaces, this entails setting up the right form of weighted finite state transducer to represent a word. Generative score-spaces, however, also include derivatives of the log-likelihoods, and require an additional trick. This paper has introduced a way of exploiting *expectation semirings* within the same framework, so that any order of derivatives can be found. This still takes amortised constant time in the length of the utterance. Using a first-order generative score-space, recognition performance increases with 7 to 30 % relative.

This paper is an initial investigation that leaves ample room for extension. Future directions will include approximations to make decoding with a larger vocabulary possible. Second-order score-spaces should benefit from more data and regularisation and sparsification. Since the approach with expectation semirings extends to the derivatives of any quantity that can be factorised along the arcs of a weighted finite state transducer, adding derivatives of other features is also possible. Training the parameters of the generative model within the log-linear model may increase the performance over the current discriminatively trained model, as may optimising segmentations while training.

A Score-spaces

The likelihood is defined as the sum of the weights of each path in WFST $\mathcal{T}$:

$$l(\lambda) = \sum_{\pi} l[\pi;\lambda]\,; \qquad\qquad l[\pi;\lambda] = \prod_{e\in\pi} l[e;\lambda] \tag{28}$$

The following equality will be useful.

$$\begin{aligned}
\nabla_\lambda l[\pi;\lambda] &= \nabla_\lambda \prod_{e\in\pi} l[e;\lambda] \\
&= (\nabla_\lambda l[e_1;\lambda]) \prod_{e\in\pi\setminus e_1} l[e;\lambda] + l[e_1;\lambda]\,\nabla_\lambda \prod_{e\in\pi\setminus e_1} l[e;\lambda] \\
&= \sum_{e\in\pi} \frac{l[\pi;\lambda]}{l[e;\lambda]} \nabla_\lambda l[e;\lambda] \\
&= l[\pi;\lambda] \sum_{e\in\pi} \nabla_\lambda \log l[e;\lambda]\,; \qquad (29) \\
\nabla_\lambda l(\lambda) &= \sum_{\pi} \nabla_\lambda l[\pi;\lambda] = \sum_{\pi} l[\pi;\lambda] \sum_{e\in\pi} \nabla_\lambda \log l[e;\lambda]\,. \qquad (30)
\end{aligned}$$

First-order features are defined as

$$\nabla_\lambda \log l(\lambda) = \frac{1}{l(\lambda)} \nabla_\lambda l(\lambda) = \sum_{\pi} \frac{l[\pi;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} \sum_{e\in\pi} \nabla_\lambda \log l[e;\lambda]\,. \tag{31}$$

The derivation for second-order features will use

$$\begin{aligned}
\nabla_{\lambda_k} \frac{l[\pi;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} &= \frac{\nabla_{\lambda_k} l[\pi;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} - l[\pi;\lambda] \frac{1}{(\sum_{\pi'} l[\pi';\lambda])^2} \nabla_{\lambda_k} \sum_{\pi'} l[\pi';\lambda] \\
&= \frac{l[\pi;\lambda] \sum_{e\in\pi} \nabla_{\lambda_k} \log l[e;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} \\
&\qquad - \frac{l[\pi;\lambda]}{(\sum_{\pi'} l[\pi';\lambda])^2} \sum_{\pi'} l[\pi';\lambda] \sum_{e'\in\pi'} \nabla_{\lambda_k} \log l[e';\lambda] \\
&= \frac{l[\pi;\lambda]}{\sum_{\pi'} l[\pi';\lambda]} \left(\sum_{e\in\pi} \nabla_{\lambda_k} \log l[e;\lambda] \right) - \nabla_{\lambda_k} \log l(\lambda)\,. \qquad (32)
\end{aligned}$$

Second-order features then are defined as

$$\begin{aligned}
\nabla_{\lambda_k}\nabla_{\lambda_j}\log l(\boldsymbol{\lambda}) &= \sum_{\pi} \nabla_{\lambda_k} \frac{l[\pi;\boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi';\boldsymbol{\lambda}]} \sum_{e\in\pi} \nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \\
&= \sum_{\pi} \Bigg(\left[\nabla_{\lambda_k} \frac{l[\pi;\boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi';\boldsymbol{\lambda}]}\right] \sum_{e\in\pi} \nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \\
&\qquad + \frac{l[\pi;\boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi';\boldsymbol{\lambda}]} \sum_{e\in\pi} \nabla_{\lambda_k}\nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \Bigg) \\
&= \sum_{\pi} \frac{l[\pi;\boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi';\boldsymbol{\lambda}]} \Bigg(\Bigg(\sum_{e\in\pi} \nabla_{\lambda_k}\log l[e;\boldsymbol{\lambda}] - \nabla_{\lambda_k}\log l(\boldsymbol{\lambda}) \Bigg) \sum_{e\in\pi} \nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \\
&\qquad\qquad + \sum_{e\in\pi} \nabla_{\lambda_k}\nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \Bigg) \\
&= \sum_{\pi} \frac{l[\pi;\boldsymbol{\lambda}]}{\sum_{\pi'} l[\pi';\boldsymbol{\lambda}]} \Bigg(\Bigg(\sum_{e\in\pi} \nabla_{\lambda_k}\log l[e;\boldsymbol{\lambda}] \Bigg) \Bigg(\sum_{e\in\pi} \nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \Bigg) \\
&\qquad\qquad + \sum_{e\in\pi} \nabla_{\lambda_k}\nabla_{\lambda_j}\log l[e;\boldsymbol{\lambda}] \Bigg) - (\nabla_{\lambda_k}\log l(\boldsymbol{\lambda}))\big(\nabla_{\lambda_j}\log l(\boldsymbol{\lambda})\big)\,.
\end{aligned} \tag{33}$$

Bibliography

Corinna Cortes, Patrick Haffner, and Mehryar Mohri (2004). "Rational Kernels: Theory and Algorithms." *Journal of Machine Learning Research* 5, pp. 1035–1062.

Jason Eisner (2002). "Parameter Estimation for Probabilistic Finite-State Transducers." In *Proceedings of the Annual Meeting on Association for Computational Linguistics*. pp. 1–8.

M. J. F. Gales and F. Flego (2010). "Discriminative classifiers with adaptive kernels for noise robust speech recognition." *Computer Speech and Language* 24 (4), pp. 648–662.

Georg Heigold, Hermann Ney, Patrick Lehnen, Tobias Gass, and Ralf Schlüter (2011). "Equivalence of Generative and Log-Linear Models." *IEEE Transactions on Audio, Speech, and Language Processing* 19 (5), pp. 1138–1148.

Hans-Günter Hirsch and David Pearce (2000). "The AURORA experimental framework for the performance evaluation of speech recognition systems under noise conditions." In *Proceedings of ASR*. pp. 181–188.

Björn Hoffmeister, Georg Heigold, Ralf Schlüter, and Hermann Ney (2012). "WFST Enabled Solutions to ASR Problems: Beyond HMM Decoding." *IEEE Transactions on Audio, Speech, and Language Processing* 20 (2), pp. 551–564.

Tommi Jaakkola and David Haussler (1998). "Exploiting Generative Models in Discriminative Classifiers." In *Proceedings of NIPS*.

Martin Layton (2006). *Augmented Statistical Models for Classifying Sequence Data*. Ph.D. thesis, Cambridge University.

Martin Layton and Mark Gales (2007). "Acoustic Modelling Using Continuous Rational Kernels." In *Journal of VLSI Signal Processing*. pp. 67–82.

Zhifei Li and Jason Eisner (2009). "First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests." In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Mehryar Mohri, Fernando Pereira, and Michael Riley (2000). "The design principles of a weighted finite-state transducer library." *Theoretical Computer Science* 231 (1), pp. 17–32.

A. Ragni and M. J. F. Gales (2011). "Structured Discriminative Models for Noise Robust Continuous Speech Recognition." In *Proceedings of ICASSP*.

A. Ragni and M. J. F. Gales (2012). "Inference Algorithms for Generative Score-Spaces." In *Proceedings of ICASSP*. p. accepted.

Sunita Sarawagi and William W. Cohen (2004). "Semi-Markov Conditional Random Fields for Information Extraction." In *Proceedings of NIPS*.

Nathan Smith and Mark Gales (2001). "Speech Recognition using SVMs." In *Proceedings of NIPS*.

Ben Taskar, Carlos Guestrin, and Daphne Koller (2003). "Rational Kernels: Theory and Algorithms." In *Proceedings of* *nips*.

Geoffrey Zweig and Patrick Nguyen (2010). "From Flat Direct Models to Segmental crf Models." In *Proceedings of* *icassp*.